Session Code: DAT401

data systems

T-SQL Enhancements in SQL Server "Yukon"

Balaji Rathakrishnan
Group Program Manager
Microsoft Corporation
balajir@microsoft.com

PDC03

Make the connection

Microsoft

Agenda

- Query language: do more in a single statement
  - Recursive queries
  - Ranking functions
  - Relational operators
  - DML with outputs
  - Fulltext search
- Richer data types
- Procedural T-SQL

Common table expressions (CTEs) and Recursive Queries

Ability to traverse recursive hierarchies in a single query

Scenarios

- Hierarchy in a table (MGRID-EMPID, Part-Subpart)
  - Find all employees reporting to a manager or
  - Find all parts required to assemble a product

Common Table Expressions

- As per SQL-99
- Common table expressions

```
WITH <CTEName> ( <column-list> )
AS
( <CTE>)
<SELECT using CTE>
```

- Both recursive and non-recursive forms
- Non-recursive:
  - Rewrite queries with derived tables to be more readable.

Recursive CTEs

- Recursive, when <CTE> references itself
- Recursive form of CTE

```
<non-recursive SELECT>
UNION ALL
<SELECT referencing CTE>
```

Initialize

Accumulate

- Recursion stops when 2nd SELECT produces empty results

Without Recursive Queries

```
DECLARE @RowsAdded int

-- table variable to hold accumulated results
DECLARE @reports TABLE (empid nchar(5) primary key, empname nvarchar(50) NOT NULL, mgrid
        nchar(5), title nvarchar(30), processed tinyint default 0)

-- initialize @Reports with direct reports of the given employee
INSERT @reports
SELECT empid, empname, mgrid, title, 0
FROM employees
WHERE empid = '12345'

SET @RowsAdded = @@rowcount

-- While new employees were added in the previous iteration
WHILE @RowsAdded > 0
BEGIN /*Mark all employee records whose direct reports are going to be found in this iteration with
        processed=1.*/
        UPDATE @reports
        SET processed = 1
        WHERE processed = 0

        -- Insert employees who report to employees marked 1.
        INSERT @reports
        SELECT e.empid, e.empname, e.mgrid, e.title, 0
        FROM employees e, @reports r
        WHERE e.mgrid=r.empid and e.mgrid <> e.empid and r.processed = 1

        SET @RowsAdded = @@rowcount
        /*Mark all employee records whose direct reports have been found in this iteration.*/
        UPDATE @reports SET processed = 2 WHERE processed = 1
END
```

With Recursive Queries

```
WITH EmpCTE(empid, empname, mgrid)
AS
(
  SELECT empid, empname, mgrid
  FROM Employees
  WHERE empid = '12345'
        UNION ALL
  SELECT E.empid, E.empname, E.mgrid
  FROM Employees AS E JOIN EmpCTE AS M
       ON E.mgrid = M.empid
)
SELECT * FROM EmpCTE
```

Ranking Functions

- Ranking functions in Yukon
  - RANK()
  - DENSE_RANK()
  - NTILE(<expression>)
  - ROW_NUMBER()
- Syntax based on SQL-99 OLAP Extensions

```
<ranking_function>
OVER( [PARTITION BY <column>]
                  ORDER BY <column>)
```

Ranking Functions: Scenarios

- Data analysis (RANK, DENSE_RANK, NTILE)
  - Ability to generate ranks based on different criteria in same query
  - Ability to separate presentation order from ranks
- Paging using ROW_NUMBER
  - Common scenario for walking through result sets

Ranking Function Scenarios:

Data analysis

Find ranking of products based on Unit sales, Revenue

```
SELECT
   RANK() OVER(ORDER BY UnitSales),
   RANK() OVER(ORDER BY Revenue),
   RANK()  OVER(ORDER BY UnitProfit)
FROM ProductSales
ORDER BY Name
```

Ranking functions, paging using row_number

demo

PDC03

Make the connection

New Relational Operators
PIVOT/UNPIVOT/APPLY

- PIVOT
  - Transforms a set of rows to columns
  - Similar to Access TRANSFORM
  - Useful for open schemas/OLAP scenarios
- UNPIVOT
  - Reverse operation of PIVOT
- APPLY
  - Allows evaluating a table-valued function for every row of outer-table

New Relational Operators
PIVOTing Name-Value pairs

ObjID	PropName	PropVal
1	Name	x.doc
1	CrDate	12/3/2001
2	Name	Sales.xls
2	Author	Mary Higgins

```
SELECT *
FROM table
  PIVOT(MIN(PropVal)
  FOR PropName IN
  ([Name],[Author])) t
```

ObjID	Name	Author
1	Spec.doc	NULL
2	Sales.xls	Mary Higgins

DML with OUTPUT

OUTPUT clause for DML

- Ability to return rows as part of DML operations
- Use “Inserted” and “Deleted” columns available to get pre- and post-update values
- Option to store returned rows
  - OUTPUT… INTO…

```
UPDATE Orders
SET status='processed'
OUTPUT DELETED.*, INSERTED.*
WHERE status='unprocessed'
```

Full-Text Search

- Fast and powerful index and query of textual data

```
SELECT ProductName
FROM Products
WHERE CONTAINS(ProductName, 'spread
                            NEAR
Boysenberry')
```

- Scalability, Performance enhancements:
  - Index building is order of magnitude faster
  - Query Performance is 30-50% faster

Full-Text Search: New Features

- Thesaurus support
- Multi-column specification on Full-Text queries

  CONTAINS((col1,col2), 'Yukon')
- Improved language support, configurable accent sensitivity/insensitivity
- Enhanced support for distributed queries:
  - Fulltext queries against linked servers
- Support for indexed views

Fulltext Search: Better Manageability

- Integrated Backup/Restore: Recovery process fully includes Full-Text catalogs
- Enhanced transportability of Full-Text catalogs through database attach/detach
- Instance-level resources (no more shared components)
- MS Search runs in same security context as SQL Server

Agenda

- Expressive power – do more in a single T-SQL statement
- Richer data types
  - XML
  - Varchar(max), nvarchar(max), varbinary(max)
- Procedural T-SQL

XML Support
Scenarios

- Platform independent data/message format that enables data and application Integration
- Query and update semi-structured data
  - Documents, email

Now supported deeply in the SQL Server database

Learn more about XML data type tomorrow: DAT 402 (10/29/03

XML data type

- First class data type in T-SQL
  - Columns, Variables, Parameters
- Optionally constrained by XML Schema
- Data Manipulation
  - XQuery (W3C Standard) + DML
  - Ability to reference columns variables in XQuery
- XML Indexes

```
CREATE TABLE Candidates (Id int,
                         Resume XML)
SELECT Id, Resume::Query('//Education')
FROM Candidates
```

Varchar(max) Type

- Extension to varchar, nvarchar, varbinary up to 2GB
  - Uses MAX size specifier

```
CREATE TABLE myTable
(Id int,
 Picture varbinary(max))
```

- Alternative to text/ntext/image
  - No text pointer support

Varchar(max) Type

- Brings together programming model of small and large values
  - Comparisons
  - Triggers
  - Concatenation
  - Aggregates
  - Parameters
  - Variables
- Facilitates smooth transition when small string/binary data outgrows 8k limit

Varchar(max) Type

- All string functions operate on varchar(max)
  - SUBSTRING used to read chunks
- UPDATE statement enhanced to support update of CHUNKS

```
UPDATE myTable
SET Picture::Write(@newchunk,
  @offset, @remove)
```

Agenda

- Expressive power – do more in a single T-SQL statement
- Richer data types
- Procedural T-SQL
  - DDL Triggers
  - Transaction abort handling
  - Performance: Statement-level recompile

DDL Triggers

- Extension of traditional triggers for DDL events
- Triggering events include all DDL statements
  - CREATE_TABLE, ALTER_PROCEDURE, DROP_LOGIN, etc.
- Scoping at Database and Server levels
- Event data available inside trigger through eventdata() function

DDL Triggers: Scenarios

- Enforce development rules/standards for objects in a DB
  - Fail CREATE/ALTER if rule is violated
- Protect from accidental drops
- Object checkin/checkout
- Source versioning
- Log management activity

DDL Triggers

demo

PDC03

Make the connection

Transaction Abort Handling: Scenarios

- Log state when errors fatal to the transaction occur
- Eliminate “if @@error” code
  - If you can run w/ XACT_ABORT ON
- Hold on to transaction after severe error
- Explicitly doom transactions using RAISERROR

Tran Abort Handling: Details

- TRY/CATCH blocks
  - Control flow shifts to CATCH on errors fatal to the transaction
  - Transaction remains in “doomed” state until explicitly rolled back
    - No actions which result in log writes may be performed in a doomed transaction
  - @@error may be queried as first statement in CATCH block
- RAISERROR … WITH TRAN_ABORT to explicitly doom transaction

Transaction Abort Handling: Example

```
CREATE PROCEDURE add_to_foo @a int, @b nvarchar(1000)
AS
BEGIN TRY
    BEGIN TRAN
    --Constraint violations cause txn/batch-abort; control to client
    INSERT foo VALUES (@a, @b)
    COMMIT TRAN
END TRY
BEGIN CATCH TRAN_ABORT
    ROLLBACK
    INSERT bad_foo VALUES (@a, @b, GETDATE())
    RAISERROR ('Logged bad insert', 17, 1) WITH TRAN_ABORT
END CATCH
```

Statement Level Recompile

- SQL Server 2000: Module-level compilation and recompilation
  - Module: sp, trigger, function, batch, etc.
- Yukon: Module-level compilation, statement level recompilation
- Recompilation due to various conditions: E.g., statistics change, changes in SET options, etc.

Statement Level Recompile

- Recompilations cost less; fewer recompilations
  - Statement level plans cached along with module level plans
- Scenarios that benefit most:
  - Recompiles caused by statistics changes
  - Batch is large and only small portion needs recompile (single statement in the case of statistics change)
  - New table is created inside the batch

T-SQL Debugging

- No more configuration steps
- Works out of the box with:
  - SQL Workbench (beta 2)
  - Visual Studio “Whidbey” (now)
- Debugging SQL client → SQL Server using Visual Studio “Whidbey”
- Seamless cross-language debugging between T-SQL and managed

Miscellaneous

- New catalog views: official metadata interface
  - Consistent, read-only access to ALL metadata
  - Sys.tables, sys.columns, …
  - Old system tables still supported for compatibility
- Permission based catalog visibility

Miscellaneous

- EXECUTE AS for procedures and functions
  - Enables executing as owner as opposed to caller
- Performance tuning/troubleshooting aids:
  - New virtual tables: sys.request_stats

T-SQL vs. .NET Framework

- T-SQL
  - Predominant data access scenarios with little business logic
  - Declarative set-based access: expressive power, efficient
  - Static embedded SQL inside procedural logic

T-SQL vs. .NET Framework

- .NET Framework
  - Server extensibility:
    - user-defined functions, types and aggregates
  - Business logic and computation with little or no data access
  - Use of .NET Frameworks APIs
  - Leverage symmetry with mid-tier programming model
- Leverage strengths of both:
  - T-SQL Queries calling managed functions

Summary

- T-SQL is alive and kicking
- Use T-SQL and .NET Framework judiciously in your applications
- Take advantage of
  - New query language enhancements
  - Improved development experience

SQL Server Resources

- Week long SQL Server “Ask the Experts” lounge in:
  - Foyer outside Room 309
- Support for SQL Server “Yukon” PDC Preview at SQLJunkies
  - http://www.sqljunkies.com/forums
- SQL Server “Yukon” FAQ Blog at SQLTeam
  - http://yukonblog.sqlteam.com/
- Other Key Resources
  - http://www.microsoft.com/sql/community
  - http://msdn.microsoft.com/sqlserver/
  - 34 world wide user groups,
    - http://msdn.microsoft.com/usergroups/find.asp
  - Our most active SQL Server newsgroups,
    - Microsoft.public.sqlserver.programming
    - Microsoft.public.sqlserver.server
    - Microsoft.public.sqlserver.dts
    - Microsoft.public.sqlserver.olap
    - Microsoft.public.sqlserver.setup
    - Microsoft.public.sqlserver.replication
    - Microsoft.public.sqlserver.msde

Microsoft®